HEX-Programs with Existential Quantification

Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl

{eiter,fink,tkren,redl}@kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Knowledge-Based Systems Group

September 13, 2013

Motivation

HEX-Programs

- Extend ASP by external sources
- Traditional safety not sufficient due to value invention
- Notion of liberal domain-expansion safety guarantees finite groundability

Example

$$\Pi = \left\{ \begin{array}{ll} r_1 : t(a). & r_3 : s(Y) \leftarrow t(X), \&cat[X,a](Y). \\ r_2 : dom(aa). & r_4 : t(X) \leftarrow s(X), dom(X). \end{array} \right\}$$

Contribution

- Domain-specific existential quantification in rule heads
- Grounding algorithm extended by application-specific termination hooks
- Instances: model computation over acyclic programs, query answering over programs with logical existential quantifier, function symbols

HEX-Programs

HEX-programs extend ordinary ASP programs by external sources

Definition (HEX-programs)

A HEX-program consists of rules of form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n,$$

with classical literals a_i, and classical literals or an external atoms b_j.

Definition (External Atoms)

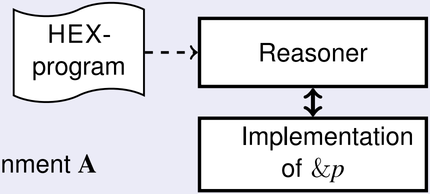
An external atom is of the form

$$\&p[q_1, \ldots, q_k](t_1, \ldots, t_l),$$

p ...external predicate name

q_i ...predicate names or constants

t_j ...terms



Semantics:

$1 + k + l$-ary Boolean oracle function $f_{\&p}$:

$\&p[q_1, \ldots, q_k](t_1, \ldots, t_l)$ is true under assignment $\mathbf{A}$ iff $f_{\&p}(\mathbf{A}, q_1, \ldots, q_k, t_1, \ldots, t_l) = 1$.

Domain-specific Existential Quantification

Idea

- Introduce new values which may appear in answer sets
- Structure of these values matters
- Introduction may be subject to constraints outside the program

Realization: Use value invention in rule body, transfer new values to the head

Example

$$iban(B, I) \leftarrow country(B, C), bank(B, N), \&iban[C, N, B](I).$$

Example

$$lifetime(M, L) \leftarrow machine(M, C), \&lifetime[M, C](L).$$

Existential Quantification

We will now discuss 3 instances of our approach:

- Model-building over acyclic $\text{HEX}^{\exists}$-Programs
- Query Answering over positive $\text{HEX}^{\exists}$-Programs
- Function Symbols

Algorithm BGroundHEX

Input: A HEX-program Π
Output: A ground HEX-program Π_g

$\Pi_p = \Pi \cup \{r_{inp}^{\&g[\vec{Y}](\vec{X})} \mid \&g[\vec{Y}](\vec{X})$ in $r \in \Pi\}$

Replace all external atoms $\&g[\vec{Y}](\vec{X})$ in all rules r in Π_p by $e_{r,\&g\vec{Y}}(\vec{X})$

while *Repeat()* **do**

- $PIT \leftarrow \emptyset$
- $NewInputTuples \leftarrow \emptyset$
- **repeat**
  - $\Pi_{pg} \leftarrow \text{GroundASP}(\Pi_p)$
  - **for** $\&g[\vec{Y}](\vec{X})$ *in a rule* $r \in \Pi$ **do**
    - $\mathbf{A}_{ma} = \{\mathbf{T}p(\vec{c}) \mid a(\vec{c}) \in A(\Pi_{pg}), p \in \vec{Y}_m\} \cup \{\mathbf{F}p(\vec{c}) \mid a(\vec{c}) \in A(\Pi_{pg}), p \in \vec{Y}_a\}$
    - **for** $\mathbf{A}_{nm} \subseteq \{\mathbf{T}p(\vec{c}), \mathbf{F}p(\vec{c}) \mid p(\vec{c}) \in A(\Pi_{pg}), p \in \vec{Y}_n\}$ s.t. $\nexists a : \mathbf{T}a, \mathbf{F}a \in \mathbf{A}_{nm}$ **do**
      - $\mathbf{A} = (\mathbf{A}_{ma} \cup \mathbf{A}_{nm} \cup \{\mathbf{T}a \mid a \leftarrow \in \Pi_{pg}\}) \setminus \{\mathbf{F}a \mid a \leftarrow \in \Pi_{pg}\}$
      - **for** $\vec{y} \in \{\vec{c} \mid r_{inp}^{\&g[\vec{Y}](\vec{X})}(\vec{c}) \in A(\Pi_{pg}) s.t.$ *Evaluate*$(r_{inp}^{\&g[\vec{Y}](\vec{X})}(\vec{c})) = true\}$ **do**
        - Let $O = \{\vec{x} \mid f_{\&g}(\mathbf{A}, \vec{y}, \vec{x}) = 1\}$
        - $\Pi_p \leftarrow \Pi_p \cup \{e_{r,\&g[\vec{y}]}(\vec{x}) \vee ne_{r,\&g[\vec{y}]}(\vec{x}) \leftarrow \mid \vec{x} \in O\}$
        - $NewInputTuples \leftarrow NewInputTuples \cup \{r_{inp}^{\&g[\vec{Y}](\vec{X})}(\vec{y})\}$
  - $PIT \leftarrow PIT \cup NewInputTuples$
- **until** Π_{pg} *did not change*

Remove input auxiliary rules and external atom guessing rules from Π_{pg}
Replace all $e_{\&g[\vec{y}]}(\vec{x})$ in Π_{pg} by $\&g[\vec{y}](\vec{x})$
return Π_{pg}

Model-building over Acyclic HEX$^{\exists}$-Programs

Definition

A HEX$^{\exists}$*-program* is a finite set of rules of form

$$\forall \vec{X} \exists \vec{Y} : \mathbf{atom}[\vec{X'} \cup \vec{Y}] \leftarrow \mathbf{conj}[\vec{X}], \quad (1)$$

where $\vec{X}$ and $\vec{Y}$ are disjoint sets of variables, $\vec{X'} \subseteq \vec{X}$, $\mathbf{atom}[\vec{X}]$.

Definition

For HEX$^{\exists}$-program Π let $T_{\exists}(\Pi)$ be the HEX-program where each

$$r = \exists \vec{Y} : \mathbf{atom}[\vec{X'} \cup \vec{Y}] \leftarrow \mathbf{conj}[\vec{X}]$$

is replaced by

$$\mathbf{atom}[\vec{X'} \cup \vec{Y}] \leftarrow \mathbf{conj}[\vec{X}], \&exists^{|\vec{X'}|,|\vec{Y}|}[r, \vec{X'}](\vec{Y}),$$

where $f_{\&exists^{n,m}}(\mathbf{A}, r, \vec{x}, \vec{y}) = 1$ iff $\vec{y} = \phi_1, \ldots, \phi_m$ is a vector of *fresh and unique null values* for $r, \vec{x}$ and do not appear in Π, and $f_{\&exists^{n,m}}(\mathbf{A}, r, \vec{x}, \vec{y}) = 0$ otherwise.

Model-building over Acyclic HEX$^{\exists}$-Programs

Example

Program Π:

$$employee(john). \quad employee(joe).$$
$$r_1 : \exists Y : office(X, Y) \leftarrow employee(X).$$
$$r_2 : room(Y) \leftarrow office(X, Y)$$

Program $T_{\exists}(\Pi)$:

$$employee(john). \quad employee(joe).$$
$$r'_1 : office(X, Y) \leftarrow employee(X), \&exists^{1,1}[r_1, X](Y).$$
$$r_2 : room(Y) \leftarrow office(X, Y)$$

The unique answer set of $T_{\exists}(\Pi)$ is
$\{employee(john), employee(joe), office(john, \phi_1),$
$office(joe, \phi_2), room(\phi_1), room(\phi_2)\}$.

Model-building over Acyclic HEX$^{\exists}$-Programs

For de-safe programs we do not need the hooks, thus let GroundDESafeHEX be the instantiation of BGroundHEX where

- Repeat repeats exactly once
- Evaluate return always *true*

Then:

Proposition

For de-safe programs Π, $\mathcal{AS}(\textit{GroundDESafeHEX}(\Pi)) \equiv^{pos} \mathcal{AS}(\Pi_g)$.

Query Answering over Positive HEX$^{\exists}$-Programs

Definition

A *Datalog*$^{\exists}$-program is a finite set of rules of form $\forall\vec{X}\exists\vec{Y} : \mathbf{atom}[\vec{X}' \cup \vec{Y}] \leftarrow \mathbf{conj}[\vec{X}]$ where $\vec{X}$ and $\vec{Y}$ are disjoint sets of variables, $\vec{X}' \subseteq \vec{X}$.
Disallowed: default negation, general external atoms.

Definition

A homomorphism is a mapping $h : \mathcal{N} \cup \mathcal{V} \to \mathcal{C} \cup \mathcal{V}$.
A homomorphism h is called substitution if $h(N) = N$ for all $N \in \mathcal{N}$.

Definition

Model of a program: set of atoms M s.t. whenever there is a substitution h with $h(B(r)) \subseteq M$ for some $r \in \Pi$, then $h|_{\vec{X}}(H(r))$ is substitutive to some atom in M.

Definition

A conjunctive query q is of form $\exists\vec{Y} :\leftarrow \mathbf{conj}[\vec{X} \cup \vec{Y}]$ with free variables $\vec{X}$.

Query Answering over Positive HEX$^{\exists}$-Programs

Answer of a CQ q with free variables $\vec{X}$ wrt. model M:

$ans(q, M) = \{h|_{\vec{X}} \mid h$ is a substitution and $h|_{\vec{X}}(q)$ is substitutive to some $a \in M\}$

Answer of a CQ q wrt. a program Π:

$ans(q, \Pi) = \{h \mid h \in ans(q, M)\ \forall M \in mods(\Pi)\}$

Query Answering over Positive HEX$^{\exists}$-Programs

Answer of a CQ q with free variables $\vec{X}$ wrt. model M:

$ans(q, M) = \{h|_{\vec{X}} \mid h$ is a substitution and $h|_{\vec{X}}(q)$ is substitutive to some $a \in M\}$

Answer of a CQ q wrt. a program Π:

$ans(q, \Pi) = \{h \mid h \in ans(q, M)\ \forall M \in mods(\Pi)\}$

Definition

Model U of a program Π is universal if, for each $M \in mods(\Pi)$, there is a homomorphism h s.t. $h(U) \subseteq M$.

Proposition

Let U be a universal model of $Datalog^{\exists}$-program Π. Then for each CQ q, $h \in ans(q, \Pi)$ iff $h \in ans(q, U)$ and $h : \mathcal{V} \to \mathcal{C} \setminus \mathcal{N}$.

Query Answering over Positive HEX$^{\exists}$-Programs

Answer of a CQ q with free variables $\vec{X}$ wrt. model M:

$ans(q, M) = \{h|_{\vec{X}} \mid h$ is a substitution and $h|_{\vec{X}}(q)$ is substitutive to some $a \in M\}$

Answer of a CQ q wrt. a program Π:

$ans(q, \Pi) = \{h \mid h \in ans(q, M)\ \forall M \in mods(\Pi)\}$

Definition

Model U of a program Π is universal if, for each $M \in mods(\Pi)$, there is a homomorphism h s.t. $h(U) \subseteq M$.

Proposition

Let U be a universal model of $Datalog^{\exists}$-program Π. Then for each CQ q, $h \in ans(q, \Pi)$ iff $h \in ans(q, U)$ and $h : \mathcal{V} \to \mathcal{C} \setminus \mathcal{N}$.

$\Rightarrow$ Key issue: Computing (finite subsets of) a universal model

Query Answering over Positive HEX$^{\exists}$-Programs

Example

Let Π be the following $Datalog^{\exists}$-program:

$$
\begin{aligned}
& person(john). \quad person(joe). \\
r_1 : \exists Y : father(X,Y) &\leftarrow person(X). \\
r_2 : \quad person(Y) &\leftarrow father(X,Y).
\end{aligned}
$$

Then $T_{\exists}(\Pi)$ is the following program:

$$
\begin{aligned}
& person(john). \quad person(joe). \\
r_1' : \quad father(X,Y) &\leftarrow person(X), \&exists^{1,1}[r_1, X](Y). \\
r_2 : \quad person(Y) &\leftarrow father(X,Y).
\end{aligned}
$$

Query Answering over Positive HEX$^{\exists}$-Programs

Input: A HEX-program $\Pi = T_{\exists}(\Pi_{\exists})$ for some $Datalog^{\exists}$-program $\Pi_{\exists}$, the count of freeze steps c_{freeze}
Output: A ground HEX-program Π_g s.t. $\mathbf{A} \in \mathcal{AS}(\Pi_g)$ is sound and complete for query answering

$\Pi_p = \Pi \cup \{r_{inp}^{\&g[\vec{Y}](\vec{X})} \mid \&g[\vec{Y}](\vec{X}) \text{ in } r \in \Pi\}$

Replace all external atoms $\&g[\vec{Y}](\vec{X})$ in all rules r in Π_p by $e_{r,\&g\vec{Y}}(\vec{X})$

for $f = 0, \ldots, c_{freeze}$ **do**

- $PIT \leftarrow \emptyset$
- $NewInputTuples \leftarrow \emptyset$
- **repeat**
  - $\Pi_{pg} \leftarrow \text{GroundASP}(\Pi_p)$
  - **for** $\&g[\vec{Y}](\vec{X})$ *in a rule* $r \in \Pi$ **do**
    - $\mathbf{A}_{ma} = \{\mathbf{T}p(\vec{c}) \mid a(\vec{c}) \in A(\Pi_{pg}), p \in \vec{Y}_m\} \cup \{\mathbf{F}p(\vec{c}) \mid a(\vec{c}) \in A(\Pi_{pg}), p \in \vec{Y}_a\}$
    - **for** $\mathbf{A}_{nm} \subseteq \{\mathbf{T}p(\vec{c}), \mathbf{F}p(\vec{c}) \mid p(\vec{c}) \in A(\Pi_{pg}), p \in \vec{Y}_n\}$ *s.t.* $\nexists a : \mathbf{T}a, \mathbf{F}a \in \mathbf{A}_{nm}$ **do**
      - $\mathbf{A} = (\mathbf{A}_{ma} \cup \mathbf{A}_{nm} \cup \{\mathbf{T}a \mid a \leftarrow \in \Pi_{pg}\}) \setminus \{\mathbf{F}a \mid a \leftarrow \in \Pi_{pg}\}$
      - **for** $\vec{y} \in \{\vec{c} \mid r_{inp}^{\&g[\vec{Y}](\vec{X})}(\vec{c}) \in A(\Pi_{pg})$ *which is not homomorphic to any* $a \in PIT\}$ **do**
        - Let $O = \{\vec{x} \mid f_{\&g}(\mathbf{A}, \vec{y}, \vec{x}) = 1\}$
        - $\Pi_p \leftarrow \Pi_p \cup \{e_{r,\&g[\vec{y}]}(\vec{x}) \vee ne_{r,\&g[\vec{y}]}(\vec{x}) \leftarrow \mid \vec{x} \in O\}$
        - $NewInputTuples \leftarrow NewInputTuples \cup \{r_{inp}^{\&g[\vec{Y}](\vec{X})}(\vec{y})\}$
  - $PIT \leftarrow PIT \cup NewInputTuples$
- **until** Π_{pg} *did not change*

Remove input auxiliary rules and external atom guessing rules from Π_{pg}
Replace all $e_{\&g[\vec{y}]}(\vec{x})$ in Π_{pg} by $\&g[\vec{y}](\vec{x})$
return Π_{pg}

Query Answering over Positive HEX$^{\exists}$-Programs

Example (ctd.)

Let Π be the following $Datalog^{\exists}$-program:

$$person(john). \quad person(joe).$$
$$r_1 : \exists Y : father(X,Y) \leftarrow person(X).$$
$$r_2 : \quad person(Y) \leftarrow father(X,Y).$$

Then $T_{\exists}(\Pi)$ is the following program:

$$person(john). \quad person(joe).$$
$$r_1' : \quad father(X,Y) \leftarrow person(X), \&exists^{1,1}[r_1, X](Y).$$
$$r_2 : \quad person(Y) \leftarrow father(X,Y).$$

For $c_{freeze} = 1 \Rightarrow$ program with single answer set
$\{person(john), person(joe), father(john, \phi_1), father(joe, \phi_2), person(\phi_1), person(\phi_2)\}$

Query Answering over Positive HEX$^{\exists}$-Programs

Example (ctd.)

Let Π be the following $Datalog^{\exists}$-program:

$$person(john). \quad person(joe).$$
$$r_1 : \exists Y : father(X,Y) \leftarrow person(X).$$
$$r_2 : \quad person(Y) \leftarrow father(X,Y).$$

Then $T_{\exists}(\Pi)$ is the following program:

$$person(john). \quad person(joe).$$
$$r_1' : \quad father(X,Y) \leftarrow person(X), \&exists^{1,1}[r_1, X](Y).$$
$$r_2 : \quad person(Y) \leftarrow father(X,Y).$$

For $c_{freeze} = 1 \Rightarrow$ program with single answer set
$\{person(john), person(joe), father(john, \phi_1), father(joe, \phi_2), person(\phi_1), person(\phi_2)\}$

Proposition

For a shy program Π, GroundDatalog$^{\exists}(\Pi, k)$ has a unique answer set which is sound and complete for answering CQs with up to k existential variables.

Function Symbols

Definition (Terms)

The set of terms $\mathcal{T}$ is defined as the least set s.t. $\mathcal{T} \supseteq \mathcal{V} \cup \mathcal{C}$ and $f \in \mathcal{C}, t_1, \dots, t_n \in \mathcal{T}$ implies $f(t_1, \dots, t_n) \in \mathcal{T}$.

For each $k \in \mathbb{N}$ two external predicates $\&compose_k$ and $\&decompose_k$ with $ar_{\mathsf{I}}(\&compose_k) = 1 + k$ and $ar_{\mathsf{O}}(\&compose_k) = 1$ and $ar_{\mathsf{I}}(\&decompose_k) = 1$ and $ar_{\mathsf{O}}(\&decompose_k) = 1 + k$.

Following [Calimeri et al., 2007],

$$f_{\&compose_k}(\mathbf{A}, f, X_1, \dots, X_k, T) = f_{\&decompose_k}(\mathbf{A}, T, f, X_1, \dots, X_k) = 1,$$

iff $T = f(X_1, \dots, X_k)$.

Note: $\&decompose_k$ supports a well-ordering

Function Symbols

Definition

Let Π be a HEX-program with function symbols. Then $T_f(\Pi)$ is the program where each $f(t_1, \ldots, t_n)$ in a rule r is recursively replaced by a new variable V.
If $f(t_1, \ldots, t_n)$ appears in $H(r)$ or in the input list of some external atom in $B(r)$, then $\&compose_n[f, t_1, \ldots, t_n](V)$ is added to $B(r)$, and otherwise $\&decompose_n[V](f, t_1, \ldots, t_n)$ is added to $B(r)$.

Example

Program Π:

$$
\begin{aligned}
q(z).\ q(y).& \\
p(f(f(X))) &\leftarrow q(X). \\
r(X) &\leftarrow p(X). \\
r(X) &\leftarrow r(f(X)).
\end{aligned}
$$

Then $T_f(\Pi)$ is:

$$
\begin{aligned}
&q(z).\ q(y). \\
&p(V) \leftarrow q(X), \&compose_1[f, X](U), \&compose_1[f, U](V). \\
&r(X) \leftarrow p(X). \\
&r(X) \leftarrow r(V), \&decompose_1[V](f, X).
\end{aligned}
$$

Conclusion

ASP Programs with External Sources

- Ordinary safety not sufficient due to value invention
- Notion of liberal domain-expansion safety guarantees finite groundability

Contribution

- Domain-specific existential quantifier in heads realized by external sources
  Advantage: Easy extensibility, e.g., data types, side constraints
- Grounding algorithm extended by application-specific termination hooks
- Instances: model building over acyclic programs, query answering with logical existential quantifier, function symbols

Future Work

- Combination of query answering with function symbols, default negation
- Model-building over programs with infinite but finitely representable models

References

Calimeri, F., Cozza, S., and Ianni, G. (2007).
External Sources of Knowledge and Value Invention in Logic Programming.
Annals of Mathematics and Artificial Intelligence, 50(3–4):333–361.

Eiter, T., Ianni, G., Schindlauer, R., and Tompits, H. (2006).
Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning.
In *3rd European Semantic Web Conference (ESWC'06)*, volume 4011 of *LNCS*, pages 273–287. Springer.

Leone, N., Manna, M., Terracina, G., and Veltri, P. (2012).
Efficiently computable datalog$^{\exists}$ programs.
In *KR*.

Syrjänen, T. (2001).
Omega-restricted logic programs.
In *6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, volume 2173 of *LNCS*, pages 267–279. Springer.